Address Translation Services
Revision 0.9

August 21, 2006

REVISION	REVISION HISTORY	DATE
0.3	Initial release draft specification	9/2005
0.5	Initial release of the final requirements	10/12/2005
0.7	Initial release of the specification methods	1/30/2006
0.9	Initial release of the final specification	6/12/2006
0.9	Incorporation of initial comments	6/29/2006
0.9	Edit and reformat	7/10/2006

PCI-SIG disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does PCI-SIG make a commitment to update the information contained herein.

Contact the PCI-SIG office to obtain the latest revision of the specification.

Questions regarding the ATS Specification or membership in PCI-SIG may be forwarded to:

Membership Services
www.pcisig.com
E-mail: administration@pcisig.com
Phone: 1-800-433-5177 (Domestic Only)
503-291-2569
Fax: 503-297-1090

Technical Support
techsupp@pcisig.com

DISCLAIMER

Contents

Figures

Tables

Preface

This specification describes the extensions required to allow PCI Express Endpoints to interact with an address translation agent (TA) in or above a Root Complex (RC) to enable translations of DMA addresses to be cached in the Endpoint. The purpose of having an Address Translation Cache (ATC) in an Endpoint is to minimize latency and to provide a scalable distributed caching solution that will improve I/O performance while alleviating TA resource pressure.

This specification must be used in conjunction with the *PCI Express Base Specification, Revision 1.1* and associated ECNs.

Document Organization

The specification is organized into four sections:

1. Introduction and Architectural Overview of the Address Translation Service (ATS) – This section covers the problem space and associated approach to solving this space including ATS operations.
2. ATS TLP Messages and Associated Semantics – This section provides a detailed discussion of the ATS TLP messages and their operational semantics.
3. ATS Invalidation Protocol – This section provides a detailed discussion of the ATS Invalidation protocol with a number of implementation notes to enable developers implementing to this specification.
4. ATS Configuration – This section provides a detailed discussion of ATS configuration, how to enable an ATC, etc.

Documentation Conventions

Capitalization

Some terms are capitalized to distinguish their definition in the context of this document from their common English meaning. Words not capitalized have their common English meaning. When terms such as "memory write" or "memory read" appear completely in lower case, they include all transactions of that type.

Register names and the names of fields and bits in registers and headers are presented with the first letter capitalized and the remainder in lower case.

Numbers and Number Bases

Hexadecimal numbers are written with a lower case "h" suffix, e.g., FFFh and 80h. Hexadecimal numbers larger than four digits are represented with a space dividing each group of four digits, as in 1E FFFF FFFFh. Binary numbers are written with a lower case "b" suffix, e.g., 1001b and 10b. Binary numbers larger than four digits are written with a space dividing each group of four digits, as in 1000 0101 0010b.

All other numbers are decimal.

Reference Information

Reference information is provided in various places to assist the reader and does not represent a requirement of this document. Such references are indicated by the abbreviation "(ref)." For example, in some places, a clock that is specified to have a minimum period of 400 ps also includes the reference information maximum clock frequency of "2.5 GHz (ref)."

Requirements of other specifications also appear in various places throughout this document and are marked as reference information. Every effort has been made to insure that this information accurately reflects the referenced document; however, in case of a discrepancy, this document takes precedence.

Implementation Notes

Implementation Notes should not be considered to be part of this specification. They are included for clarification and illustration only.

Terms and Acronyms

Address Translation and Protection Table (ATPT)	The data structure(s) accessed by a Translation Agent to determine the mapping of untranslated DMA addresses into translated addresses. The data structure may contain fields that indicate the protection attributes of the translation entry.
Address Translation Cache (ATC)	A hardware entity that stores recently used address translations. This term is used instead of Translation Look-aside Buffer (TLB) to differentiate the TLB used for I/O from the TLB used by the CPU. Each TA is expected to have an ATC co-located with it, but an ATC need not be co-located with a TA.
Address Translation Services (ATS)	The set of configuration, wire protocol, ATC, etc., required to deliver an address translation solutionn.
Clear	A bit with the value of 0b or the act of causing a bit to have the value of 0b.
PCIe	PCI Express
RP	PCIe Root Port – See the *PCI Express Base Specification* for additional details.
Set	A bit with the value of 1b or the act of causing a bit to have the value of 1b.
Smallest Translation Unit (STU)	The minimum increment for translation and invalidation. This value is expressed in terms of 4096-byte units and is programmed into a configuration register on the Function.
TC	PCIe Traffic Class – See the *PCI Express Base Specification* for additional details.
Translated Address	An address formed by using the results of an address Translation Request.
Translation Agent (TA)	A logical entity that converts addresses expressed in terms of one address space into an address in a different address space. A Translation Agent (TA) is associated with memory that contains translation tables that the TA will reference for address translation (see ATPT). A TA may contain an Address Translation Cache. A TA may be implemented either in hardware, software, or a combination of both. A TA is treated as implementation-specific and, therefore, is outside the scope of this specification.
Untranslated Address	An address that is formed using the existing programming mechanisms of PCIe.

1

1. Architectural Overview

Most contemporary system architectures make provisions for translating addresses from DMA (bus mastering) I/O Functions. In many implementations, it has been common practice to assume that the physical address space seen by the CPU and by an I/O Function is equivalent. While in others, this is not the case. The address programmed into an I/O Function is a 'handle' that is processed by the Root Complex (RC). The result of this processing is often a translation to a physical memory address within the central complex. Typically, the processing includes access rights checking to insure that the DMA Function is allowed to access the referenced memory location(s).

The purposes for having DMA address translation vary and include:

- Limiting the destructiveness of a 'broken' or miss-programmed DMA I/O Function
- Providing for scatter/gather
- Ability to redirect message-signaled interrupts (e.g., MSI or MSI-X) to different address ranges without requiring coordination with the underlying I/O Function
- Address space conversion (32-bit I/O Function to larger system address space)
- Virtualization support

Irrespective of the motivation, the presence of DMA address translation in the host system has certain performance implications for DMA accesses.

Depending on the implementation, DMA access time can be significantly lengthened due to the time required to resolve the actual physical address. If an implementation requires access to a main-memory-resident translation table, the access time can be significantly longer than the time for an un-translated access. Additionally, if each transaction requires multiple memory accesses (e.g., for a table walk), then the memory transaction rate (i.e., overhead) associated with DMA can be high.

To mitigate these impacts, designs often include address translation caches in the entity that performs the address translation. In a CPU, the address translation cache is most commonly referred to as a translation look-aside buffer (TLB). For an I/O TA, the term address translation cache or ATC is used to differentiate it from the translation cache used by the CPU.

While there are some similarities between TLB and ATC, there are important differences. A TLB serves the needs of a CPU that is nominally running one thread at a time. The ATC, however, is generally processing requests from multiple I/O Functions, each of which can be considered a separate thread. This difference makes sizing an ATC difficult depending upon cost models and expected technology re-use across a wide range of system configurations.

The mechanisms described in this specification allow an I/O Endpoint to participate in the translation process and provide an ATC for its own memory accesses. The benefits of having an ATC within an Endpoint include:

- Ability to alleviate TA resource pressure by distributing address translation caching responsibility (reduced probability of 'thrashing' within the TA)
- Enable ATC Endpoints to have less performance dependency on a system's ATC size
- Potential to insure optimal access latency by sending pre-translated requests to central complex.

This specification will provide the interoperability that allows PCIe Endpoints to be used in conjunction with a TA, but the TA and its Address Translation and Protection Table (ATPT) are treated as implementation-specific and are outside the scope of this specification. While it may be possible to implement ATS within other PCIe Components, this specification is confined to PCIe Endpoints and PCIe Root Complex Integrated Endpoints.

Figure 1-1 illustrates an example platform with a TA and ATPT, along with a set of PCIe Endpoints and RC Integrated Endpoints with integrated ATC. A TA and an ATPT are implementation-specific and can be distinct or integrated components within a given system design.

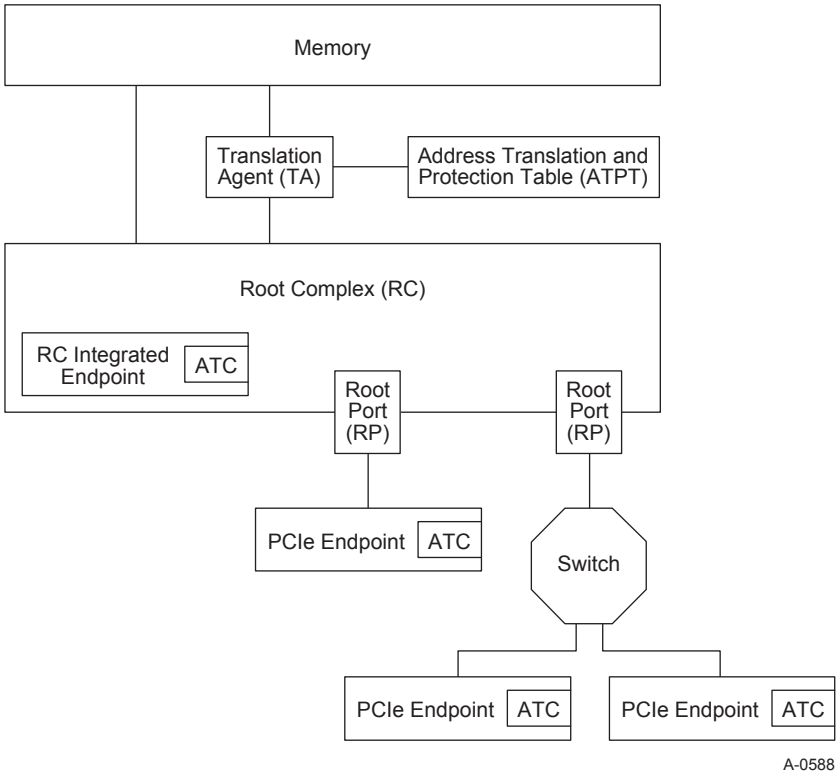

Figure 1-1: Example Illustrating a Platform with TA, ATPT, and ATC Elements

1.1. Address Translation Services (ATS) Overview

ATS builds upon the *PCI Express Base Specification* to provide a new set of TLP and associated semantics. ATS uses a request-completion protocol between an Endpoint[1] and a Root Complex (RC) to provide translation services. In addition, a new AT field is defined within the Memory Read and Memory Write TLP as defined within the *PCI Express Base Specification*. The new AT field enables an RC to determine whether a given request has been translated or not via the ATS protocol.

Figure 1-2 illustrates the basic flow of an ATS Translation Request operation.

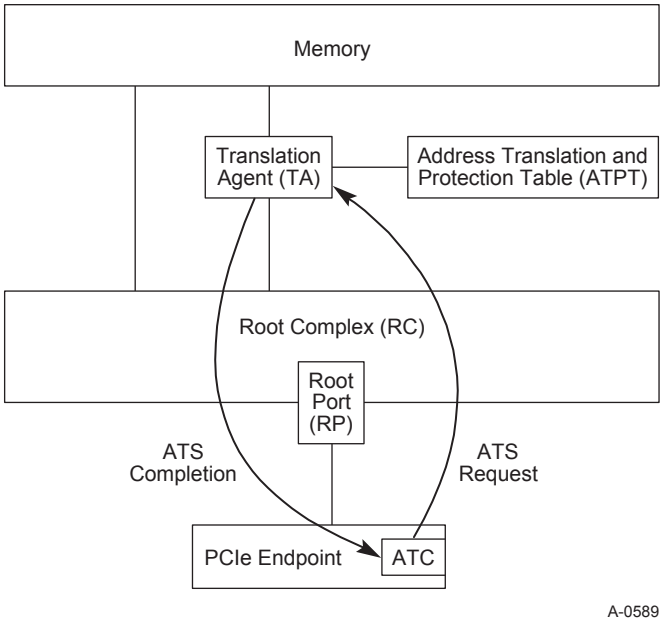

Figure 1-2: Example ATS Translation Request/Completion Exchange

In this example, a Function-specific work request is received by a single-Function PCIe Endpoint. The Function determines through an implementation-specific method that caching a translation within its ATS would be beneficial. There are a number of considerations a Function or software can use in making such a determination; for example:

- Memory address ranges that will be frequently accessed over an extended period of time or whose associated buffer content is subject to a significant update rate
- Memory address ranges, such as work and completion queue structures, data buffers for low-latency communications, graphics frame buffers, host memory that is used to cache Function-specific content, and so forth

[1] All references within this specification to an Endpoint apply equally to a PCIe Endpoint or a Root Complex Integrated Endpoint. ATS does not delineate between these two types of Endpoints in terms of requirements, semantics, configuration, error handling, etc. From a software perspective, an ATS-capable Root Complex Integrated Endpoint must behave the same as an ATS-capable non-integrated Endpoint.

Given the variability in designs and access patterns, there is no single criteria that can be applied.

The Function generates an ATS Translation Request which is sent upstream through the PCIe hierarchy to the RC which then forwards it to the TA. An ATS Translation Request uses the same routing and ordering rules as defined within the *PCI Express Base Specification*. Further, multiple ATS Translation Requests can be outstanding at any given time; i.e., one may pipeline multiple requests on one or more TC. Each TC represents a unique ordering domain and defines the domain that will be used by the associated ATS Translation Completion.

Upon receipt of an ATS Translation Request, the TA performs the following basic steps:

1. Validates that the Function has been configured to issue ATS Translation Requests.
2. Determines whether the Function may access the memory indicated by the ATS Translation Request and has the associated access rights.
3. Determines if whether a translation can be provided to the Function. If yes, the TA updates the state associated with the translation to reflect that the Function's ATC will now contain a translation for the requested address range.
    a. ATS is required to support a variety of page sizes to accommodate a range of ATPT and processor implementations.
        i. Page sizes are required to be a power of two and naturally aligned.
        ii. The minimum supported page size is 4096 bytes. ATS capable components are required to support this minimum page size.
    b. A Function will be informed of the minimum translation or invalidate size it will be required to support to provide the Function an opportunity to optimize its resource utilization. The smallest minimum translation size will be 4096 bytes.
4. The TA communicates the success or failure of the request to the RC which generates an ATS Translation Completion and transmits via a Response TLP through a RP to the Function.
    a. An RC is required to generate at least one ATS Translation Completion per ATS Translation Request; i.e., there is minimally a 1:1 correspondence independent of the success or failure of the request.
        i. A successful translation can result in one or two ATS Translation Completion TLPs per request. The Translation Completion indicates the range of translation covered.
        ii. An RC may pipeline multiple ATS Translation Completions; i.e., an RC may return multiple ATS Translation Completions and these ATS Translation Completions may be in any order relative to ATS Translation Requests.
        iii. The RC is required to transmit the ATS Translation Completion using the same TC (Traffic Class) as the corresponding ATS Translation Request.
    b. The requested address may not be valid. The RC is required to issue a Translation Completion with at least one Translation Data Value indicating that the requested address is not accessible.
5. The Function receives the ATS Translation Completion and either updates its ATC to reflect the translation or notes that a translation does not exist. The Function proceeds with processing its

work request and generates subsequent requests using either a translated address or an untranslated address based on the results of the Completion.

a. Similar to Read Completions, a Function is required to allocate resource space for each completion(s) without causing backpressure on the PCIe Link.

b. A Function is required to discard Translation Completions that might be "stale." Stale Translation Completions can occur for a variety of reasons.

As one can surmise, ATS Translation Request and Translation Completion processing is conceptually and, in many respects, identical to PCIe Read Request and Read Completion processing. This is intentional to reduce design complexity and to simplify integration of ATS into existing and new PCIe-based solutions. Keeping this in mind, ATS requires the following:

- ATS capable components must interoperate with *PCI Express Base Specification, Revision 1.1* compliant components.
- ATS is enabled through a new Capability and associated configuration structure. To enable ATS, software must detect this Capability and enable the Function to issue ATS TLP. If a Function is not enabled, the Function is required not to issue ATS Translation Requests and is required to issue all DMA Read and Write Requests with the TLP AT field set to "untranslated."
- ATS TLP are routed using either address-based and Requester ID (RID) routing.
- ATS TLP are required to use the same ordering rules as specified within the *PCI Express Base Specification*.
- ATS TLP are required to flow unmodified through PCIe 1.1-compliant Switches.
- A Function is permitted to intermix translated and untranslated requests.
- ATS transactions are required not to rely upon the address field of a memory request to communicate additional information beyond its current use as defined by the PCI-SIG.

IMPLEMENTATION NOTE

Address Range Overlap. It is likely that the untranslated and translated address range will overlap, perhaps in their entirety. This is not a requirement of ATS but may be an implementation constraint on the TA so that memory requests will be properly routed.

In contrast to the prior example, Figure 1-3 illustrates an example multi-Function Endpoint. In this example Endpoint, there are three Functions. Key points to note in Figure 1-3 are:

- Each ATC is associated with a single Function. Each ATS-capable Function must be able to source and sink at least one of each ATS Translation Request or Translation Completion type.
- Each ATC is configured and accessed on a per Function basis. A multi-Function Endpoint is not required to implement ATS on every Function.
- If the ATC implementation shares resources among a set of Functions, then the logical behavior is required to be consistent with fully independent ATC implementations.

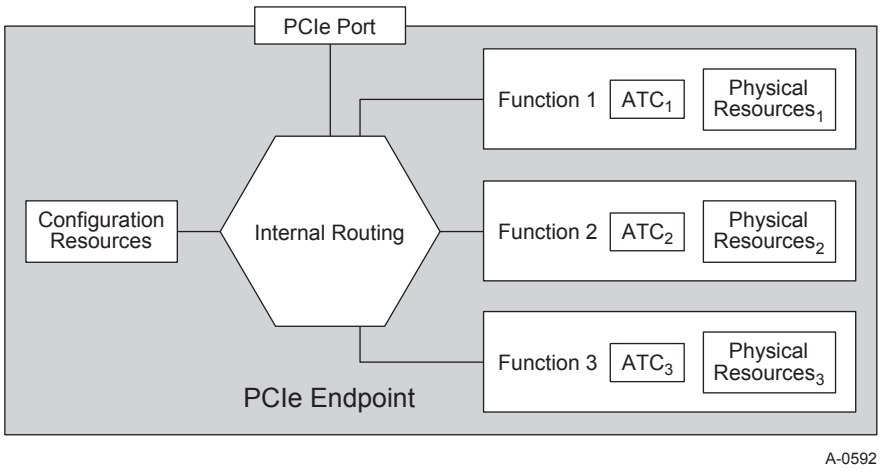

Figure 1-3: Example Multi-Function Endpoint with ATC per Function

Independent of the number of Functions within an Endpoint, the following are required:

- A Function is required not to issue any TLP with the AT field set unless the address within the TLP was obtained through the ATS Translation Request and Translation Completion protocol.
- Each ATC is required to only be populated using the ATS protocol; i.e., each entry within the ATC must be filled via an ATS Translation Completion in response to the Function issuing an ATS Translation Request for a given address.

- Each ATC cannot be modified except through the ATS protocol. That is:
  - Host system software cannot modify the ATC other than through the protocols defined in this specification except to invalidate one or more translations in an ATC. An Endpoint or Function reset would be an example of an operation preformed by software to change the contents of the ATC, but a reset is only allowed to invalidate entries not modify their contents.
  - It must not be possible for host system software to use software executing on the Endpoint to modify the ATC.

When a TA determines that a Function should no longer maintain a translation within its ATC, the TA initiates the ATS invalidation protocol. The invalidation protocol consists of a single Invalidation Request and one or more Invalidation Completions.

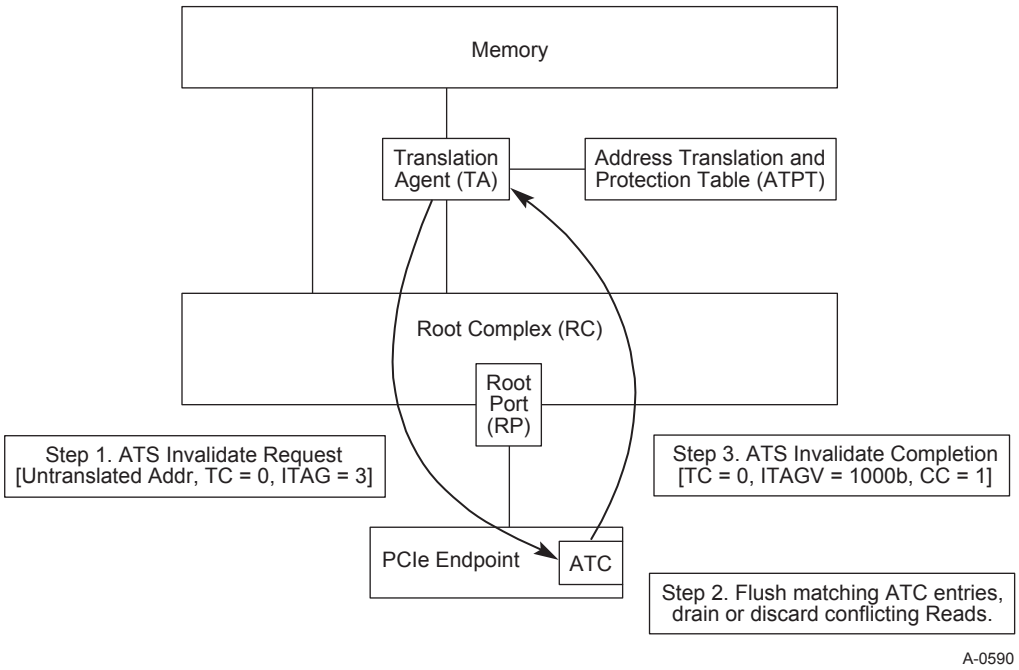

Figure 1-4: Invalidation Protocol with a Single Invalidation Request and Completion

As Figure 1-4 illustrates, there are essentially three steps in the ATS Invalidation protocol:

1. The system software updates an entry in the tables used by the TA. After the talble is changed, the TA determines that a translation should be invalidated in an ATC and initiates an Invalidation Request TLP which is transmitted from the RP to the example single-Function Endpoint. The Invalidate Request communicates an untranslated address range, the TC, and a RP unique tag which is used to correlate Invalidate Completions with the Invalidation Request.
2. The Function receives the Invalidate Request and flushes all matching ATC entries. A Function is not required to immediately flush all pending requests upon receipt of an Invalidate Request. If transactions are in a queue waiting to be sent, it is not necessary for the Function to expunge requests from the queue even if those transactions use an address that is being invalidated.

a. A Function is required not to indicate the invalidation has completed until all outstanding Read Requests or Translation Requests that reference the associated translated address have been retired or nullified.

b. A Function is required to insure that the invalidation completion indication to the RC will arrive at the RC after any previously posted writes that use the 'stale' address.

3. When a Function has ascertained that all uses of the translated address are complete, it issues one or more ATS Invalidate Completions.

   a. An Invalidate Completion is issued for each TC that may have referenced the range invalidated. These completions act as a flush mechanism to insure the hierarchy is cleansed of any in-flight transactions which may contain references to the translated address.

      i. The number of Completions required is communicated within each Invalidate Completion. A TA or RC implementation can maintain a counter to insure that all Invalidate Completions are received before considering the translation to no longer be in use.

      ii. If more than one Invalidation Complete is sent, the Invalidate Complete sent in each TC must identical.

   b. An Invalidate Completion contains the ITAG from Invalidate Request to enable the RC to correlate Invalidate Requests and Completions.

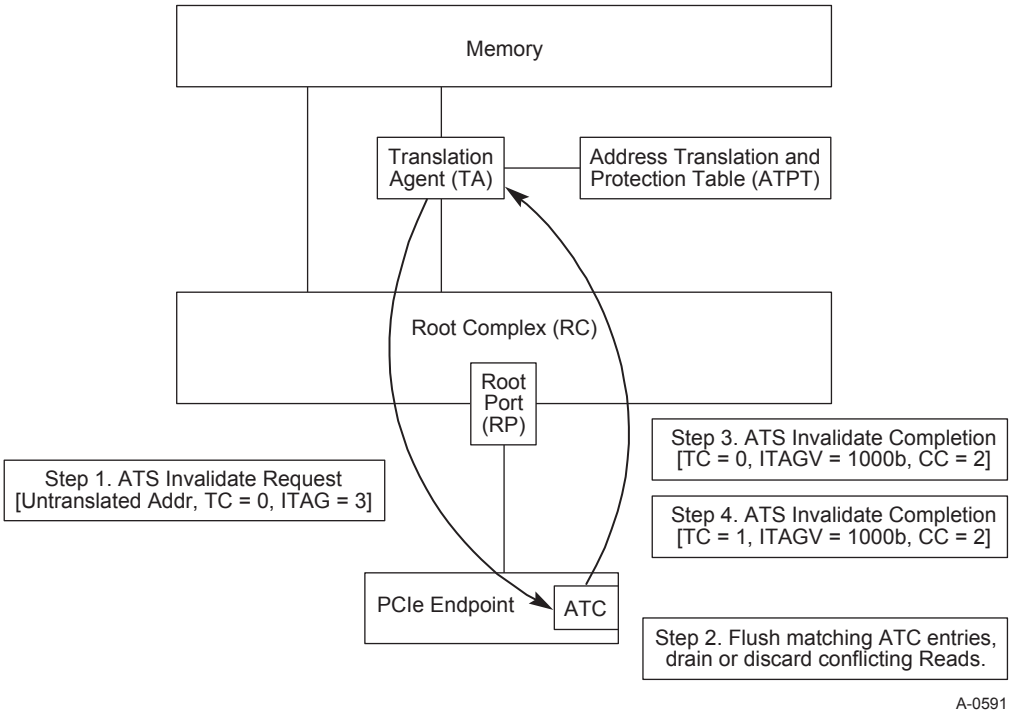

Figure 1-5: Single Invalidate Request with Multiple Invalidate Completions

2

2. ATS Translation Services

A TA does translations. An ATC can cache those translations. If an ATC is separated from the TA by PCIe, the memory request from an ATC will need to be able to indicate if the address in the transaction is translated or not. The modifications to the memory transactions are described in this section, as are the transactions that are used to communicate translations between a remote ATC and a central TA.

2.1. Memory Requests with Address Type

A Function with an ATC can send memory read/write Requests that contain either translated or un-translated addresses. As shown in Figure 2-1 and Figure 2-2, the Address Type (AT) field is used to indicate the type of address that is present in the request header.

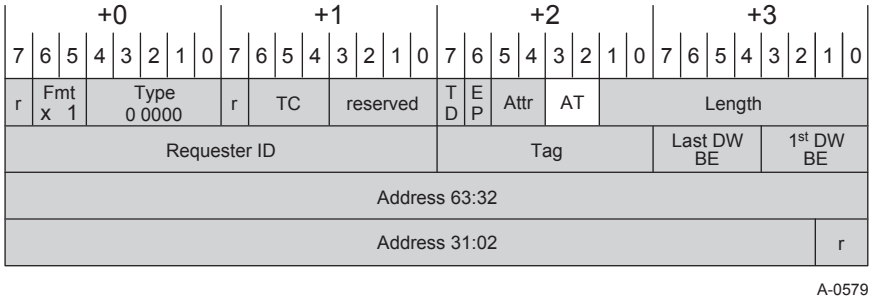

Figure 2-1: Memory Request Header with 64-bit Address

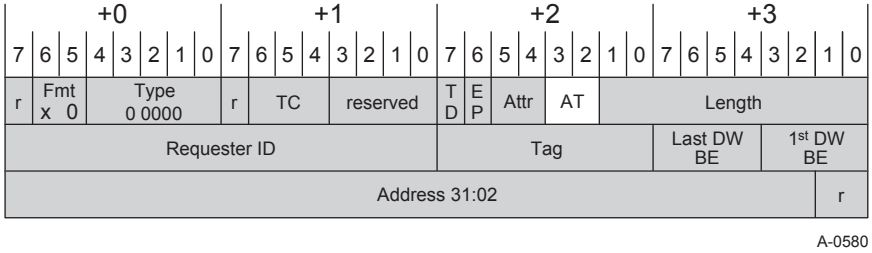

Figure 2-2: Memory Request Header with 32-bit Address

The AT field in the requests is a redefinition of a reserved field in the *PCI Express Base Specification*. Functions that do not implement an ATC will continue to set the AT field to its defined reserved value (0). Functions that implement an ATC will set the AT field as listed in Table 2-1.

Table 2-1: Address Type (AT) Field Encodings

AT Coding	Mnemonic	Meaning
00b	Default	A TA may treat the address as either virtual or physical.
01b	Translation Request	The TA will return the translation of the address contained in the address field of the request as a read completion. This value only has meaning for an explicit Translation Request (see Section 2.2) and will result in a UR if not in a Translation Request.
10b	Translated	The address in the transaction has been translated by an ATC. If the Function associated with the SourceID is allowed to present physical addresses to the system memory, then the TA may not translate this address. If the Function is not allowed to present physical addresses, then the TA may treat this as a UR.
11b	reserved	Results in a UR if used in a Read or Write Request.

2.2. Translation Requests

A Translation Request has a format that is similar to that of a memory read. The AT field is used to differentiate a Translation Request from a normal memory read.

The request header for a Translation Request has the format illustrated in Figure 2-3.

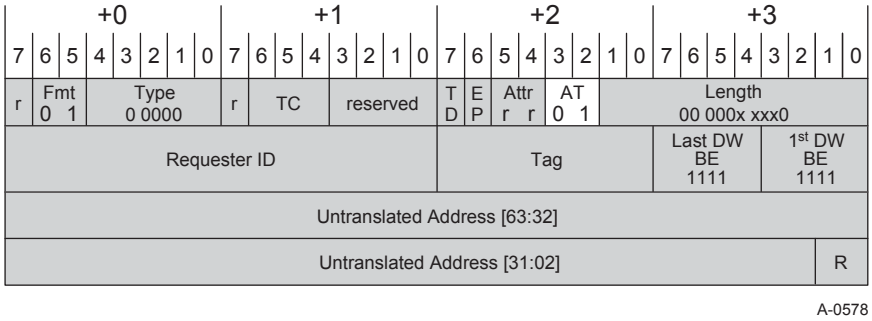

Figure 2-3: Translation Request Header

Translation Requests have the same completion timeout intervals as Read Requests.

2.2.1. Attribute Field

For a Translation Request, the Attr field is reserved for future use, and the Request will be treated as a Malformed Packet if it is not set to 00b. There are no ordering requirements for a Translation Request. A TA may reorder a Translation Request with respect to any other request.

IMPLEMENTATION NOTE

Translation Request OrderingBecause no ordering can be assumed between Translation Requests and other types of Requests, a Translation Request does not make an effective flushing/ordering primitive.

2.2.2. Length Field

The Length field is set to indicate how many translations may be returned in response to this request. Each translation is 8 bytes in length and represents one or more STUs. The maximum setting for the Length field is the lesser of the sizes determined by the setting of RCB and Max_Read_Request_Size. The Length field in a Translation Request must always indicate an even number of Dwords. If Length is set to indicate a value greater than allowed or if the least-significant bit of the Length field is non-zero, then the TA will treat the request as a Malformed Packet.

If the Length field has a value greater than two, then the Function is requesting translations for a range of memory greater than a single STU. The additional translations, if provided, will be for sequentially-increasing, equal-sized, STU-aligned regions, starting at the requested address.

2.2.3. Tag Field

The Tag field has the same meaning as in a Read Request. The Tag values used in Translation Requests come from the same pool of numbers used for Read Requests.

2.2.4. Untranslated Address Field

A Translation Request includes a 64-bit Address field. This address indicates the address to be translated. The TA will make decisions about the validity of the request, based on the address in the translation request, not the implied subsequent addresses. The TA may return fewer translations than requested, but it will not return more.

When multiple translations are requested, the TA will not return a translation if the range of that translation does not overlap the implied range of the Translation Request (this would only apply to translations after the initial value). The implied range of the Translation Request is (STU * (Length/2)).

The Untranslated Address Field in the Translation Request is any address in the range of the first STU. The TA will ignore the low-order bits that are not required to determine the translation.

The Address field in the Translation Request is shown with 52 significant bits. The number of significant bits is actually determined by the setting of the STU configuration parameter. If STU is set to 0 indicating 4096-byte increments, then only bits 63:12 are significant. If the STU is set to indicate a larger size, then the number of significant bits of the address is adjusted accordingly. The bits in the Address field that are not significant are ignored by the TA and it is not a requirement that they be set to zero.

2.3. Translation Completion

A Translation Completion (either a Cpd or a CplD) is sent by a TA for each Translation Request. This specification describes the meaning of fields in Translation Completions. Fields not defined in this specification have the same meanings proscribed for Read Completions in the *PCI Express Base Specification*.

If the TA was not able to perform the requested translation, a completion with the format shown in

Figure 2-4 is used.

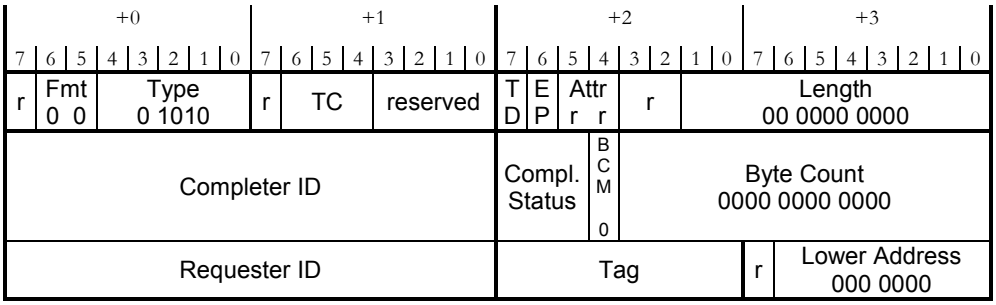

Figure 2-4: Translation Completion with No Data

The values and meaning for the Completion Status field are listed in Table 2-2.

Table 2-2: Translation Completion with No Data Status Codes

Value	Status	Meaning
000b	Success	This completion status has a nominal meaning of "success." The TA will not return this value in a Cpl.
001b	Unsupported Request (UR)	Translation Requests from this Function are not supported by the TA. If an ATC receives this Completion code, it must disable its ATC and not send requests using translated addresses until the ATC is re-enabled. This may also cause an Error Message to be sent if Advanced Error Reporting is supported.
010b	CRS	This value is not allowed in any Completion to a request initiated by a PCI Express Function. If received by a Function, it shall be treated as a malformed packet.
100b	Completer Abort (CA)	The TA was not able to translate the address because of an error in the TA. This nominally causes an error to be reported to the device driver associated with the ATC.
All others	Reserved	If a Translation Completion contains a reserved value in the Completion Status field, the Function will treat it as a malformed packet.

Note: Return values other than *Success* indicate an error.

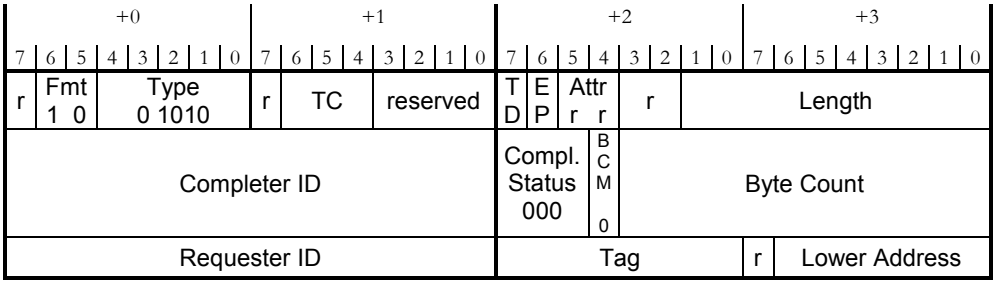

Figure 2-5: Successful Translation Completion

Fields are set in accordance with Sections 2.2.9 and 2.3.1 of the *PCI Express Base Specification*.

Translation Completions will be sent using the same TC as the Translation Request.

The Lower Address field will contain a value that will make the packet consistent with RCB semantics. If the result is returned in a single packet, Lower Address will be set to RCB minus Byte Count. If the results are returned in multiple packets, the first packet will have a Lower Address field of RCB minus (Length * 4) and subsequent packets will have a Lower Address field of 00 0000b.

If the translation is successful, the Completion Status field will be 000b and a data payload will follow the header. The contents of the data payload are shown in Figure 2-6.

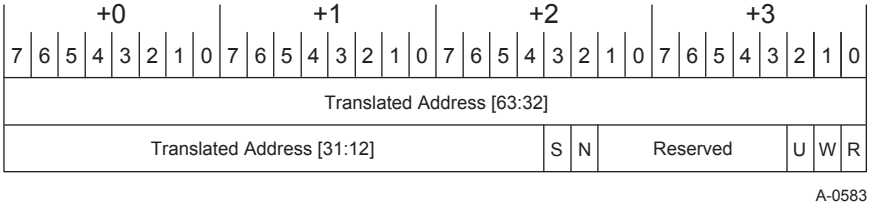

Figure 2-6: Translation Completion Data Entry

Table 2-3: Translation Completion Data Fields

Field	Meaning
S	**Size of translation** – This field is 0b if the translation applies to a 4096 byte range of memory. If this field is 1b, then the translation applies to a range of memory that is larger than 4096 bytes (see Section 2.3.1).
N	**Non-snooped accesses** – If this field is 1b, then the read and write requests that use this translation must not set the No Snoop bit in the Attribute field. If it is 0b, then the Function may use other means to determine if No Snoop should be set.
Reserved	These bits shall be ignored by the ATC.
U	**Untranslated access only** – When this field is set to 1b in a Translation Completion entry, the indicated range may only be accessed using untranslated addresses, and the Translated Address field of this Translation Completion entry may not be used in a subsequent Read/Write Request with AT set to *Translated*. This value may be cached if R or W is set to 1b.
R,W	**Read, Write** – These two fields indicate the transaction types that are allowed for the requests using the translation. The encodings are: 00b Neither read nor write transactions are allowed. This translation is considered not to be valid. The contents of the Translated Address, N, and U fields are undefined. A translation with this value may not be cached in the ATC. 01b Write Requests that target this range are allowed but Read Requests are not. 10b Read Requests that target this range are allowed but Write Requests are not. 11b Read and Write Requests that target this range are allowed.

2.3.1. Translated Address Field

If the R and W fields are both Clear, or if U is set, then the Translated Address field may not be used by the Function for any purpose.

If either the R or W field is Set, and the U field is Clear, then the Translated Address Field contains an address that can be used by the Function in a Memory Request with the AT field set to Translated; and the Function may cache the Translated Address. When cached the R and W fields must be stored with the same value as the Translation Completion entry. The address that is cached must be a subset of the address range indicated in the Translation Completion (the subset may include the entire range).

While the Translated Address is cached in the Function's ATC, it shall not be possible for the Function to modify the entry other than to delete it. The entry must be deleted from the ATC when an Invalidation Request is received that has an indicated range that overlaps any portion of the cached address.

While the E field of a Function's ATS Capability register is Set, a Function is not allowed to make an entry into its ATC unless the entry is in a Translation Completion. Entries in an ATC cache that are written before the E field is Set must not be used in Memory Request. They must either be invalidated when the E field is Set or ignored and not used.

2.3.2. Translation Range Size (S) Field

If S is set, then the translation applies to a range that is larger than 4096 bytes. If S = 1b, then bit 12 of the Translated Address is used to indicate whether or not the range is larger than 8192 bytes. If bit 12 is 0b, then the range size is 8192 bytes but it is larger than 8192 bytes if set to 1b. If S = 1b and bit 12 = 1b, then bit 13 is used to determine if the range is larger than 16384 bytes or not. If bit 13 is 0b, then the range size is 16384 bytes, but it is larger than 16384 bytes if set to 1b.

Low-order address bits are consumed in sequence to indicate the size of the range associated with the translation.

> Note: This encoding method is also used to indicate the size of the memory range being invalidated.

Examples for different translation sizes are shown in Table 2-4.

Table 2-4: Examples of Translation Size Using S Field

Address Bits																					S	Translation Range Size in Bytes
63:32*	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12		
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	4 K
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	8 K
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	1	16 K
x	x	x	x	x	x	x	x	x	x	x	x	0	1	1	1	1	1	1	1	1	1	2 M
x	x	x	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1 G
x	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4 G

*Note: Upper address bits are used to indicate the size for ranges larger than 4 GB.

The size field is set to indicate the range size in multiples of 4096 bytes regardless of the setting of STU. For example, if STU is set to indicate that the minimum translation is 8192 bytes, then S should be set to 1b on all translation returned in a Translation Completion and in all Invalidate Requests. If STU is set to indicate a 16384 bytes minimum, then S and bit 12 would both be set to 1b in all translation and invalidate ranges.

2.3.3. Non-snoop (N) Field

This field is set to 1b to indicate that Read and Write Requests that target memory in the range of this translation must not set the No Snoop Attribute bit in the Request header. When this field is 0b, the Function is allowed to set the No Snoop Attribute bit in a Function-specific manner.

Note: When this field is 0b, the Function is not allowed to set No Snoop in a Memory Request if the Enable No Snoop field in the Device Control Register is not Set.

When U is set to 1b, this meaning of this field is undefined and the TA may set this field to any value.

2.3.4. Untranslated Access Only (U) Field

This field is set to 1b when the Function is not allowed to access the implied range of memory using a translated address (the range is implied by the untranslated address in the Translation Request and the offset of the translation in the Translation Completion). The Function may use untranslated addresses to access the range as long as the accesses are allowed by the R and W fields. The Function may cache this translation value if either R or W is set to 1b. If the U bit is set, the Translated Address field in the translation is not necessarily a valid memory address and the Function may not use the value in a Read or Write Request with AT set to Translated.

Note: One of the possible uses of this field is to avoid unnecessary invalidations. If a Function uses translated requests for some portions of memory, but not others, then the U bit can be used on the portions for which translated requests are not used. When a translation changes if the U bit is set, then it will not necessarily be required that an Invalidate Request be sent to the Function. An example of this use is a Function with a ring buffer that is used for commands. The ring buffer may be allocated for a long period of time and have very high re-use (locality). For this reason, it is useful for the Function to use translated addresses in its memory request that target the command buffer. The same Function might access data buffers that have poor locality and low reuse. Accesses to the data buffers might best be handled by using untranslated Requests. Setting the U bit for the data buffer translations insures that the Function will not attempt to use a translated value to access the data buffer so, when the data buffer mappings are changed, no Invalidation Request is required. Read (R) and Write (W) Fields

These fields indicate if the returned translation value may be used in a read or write memory request. The ATC may not issue a read request using the translation value if the R bit is not set to 1b. The ATC may not issue a write request using the translation value if the W bit is not set to 1b. The ATC may not issue any type of request using the translation value if neither the R nor W bits are set to 1b. If both R and W bits are set to 0b, the range of the translation is still indicated, but the meaning of the other values in the translation is undefined.

Note: The range of a Translation entry is indicated even if R = W = 0b in order to allow a 'hole' in the Translation Completion. For example, if the Translation Request has a Length of six Dwords, then up to three translations could be included in the Translation Completion. The first and third translations may have R or W of 1b but the second could have R = W = 0b. To avoid ambiguity about the size of the indicated gap, the range of the gap is indicated in the Translation Completion even if R = W = 0b.

The R = 0b, W = 0b state is used to indicate that the address field in the translation may not be used to form a translated address value for a subsequent request.

When the host changes the translation in the TA, to make the translation present, the host is not required to send an invalidation indication to the ATC so that it will know of the change in state of the translation. Since the ATC may not be notified of changes of the translation, a translation value of R = W = 0b may not be cached.

If no table entry is found for the requested address, the TA will return a CplD with a single translation value with R = W = 0b.

> Note: Implemenations should not assume that receiving a translation response with the R or W bits set (independent of the value of the U bit) implies that a subsequent read or write request with the same **untranslated** address will succeed. Although it may be possible for a device and its controlling software to ensure this property, the method for doing so is outside this specification

2.4. Completions with Multiple Translations

An ATC is allowed to request that the TA provide translations for a virtually contiguous range of addresses. It does this by setting the Length field in the Translation Request to a value that is two times the number of requested translations as long as the request size (Length * 4) is not larger than either Max_Read_Request_Size or RCB.

If multiple translations are requested, the TA may return one or more translations as long as the number of translations does not exceed the number of requested translations. It is not an error for the TA to return fewer translations than requested and no error indication is sent unless there is an error in accessing the data.

If the Translation Completion contains multiple translations, all translations must have the same indicated size. Also, successive translation must apply to the virtual address range that abuts the previous translation in the same completion.

If a translation has both R = 0b and W = 0b, the TA must still set the Size field to the appropriate value.

Each translation in a Translation Completion will have some overlap with the implied memory range of the Translation Request (see Section 2.2).

A Translation Completion may require one or two CplDs.

If a Translation Completion CplD has a Byte Count that is greater than four times the Length field, then additional CplDs are required to complete the transaction.

If a Translation Completion CplD has a Byte Count that is equal to four times the Length field, then the packet completes the request. For such a CplD, if the sum of Byte Count and Lower Address is not a multiple of RCB, then the CplD is the last of a sequence and it is an error if no previous CplD has been received, and all translation values should be discarded.

> Note: There are multiple reasons that the TA may truncate the results of the completion. For example, the request might ask for a range of addresses, not all of which are defined. This could occur if the first translation were valid but was located at the end of a page of translations. The TA, in looking up the next page of translations, may find that the page is

not valid so the addresses are not valid. The range of addresses that are valid would be returned and no error indicated.

Note: There are multiple reasons that the TA may break a Translation Completion into multiple TLPs. As an example, if the virtual address of the Translation Completion resolves to a table access that crosses an RCB boundary of the memory system, the completion to the TA may be broken into multiple completions by the memory. Rather than require that the TA accumulate the results, it is allowed to send each portion of the Translation Completion to a Function when it is received from the system memory.

3

3. Invalidation

ATS uses the messages shown in this section to maintain consistency between the TA and the ATC. This specification assumes there is a single TA associated with each ATC. The TA (in conjunction with its associated software) must ensure that the address translations cached in the ATC are not stale by issuing Invalidate Requests.

3.1. Invalidate Request

When a translation is changed in the TA and that translation might be contained within an ATC in a Function, the TA (in conjunction with its associated software) must send an Invalidate Request to the ATC to maintain proper synchronization between the ATPT and the ATC. An Invalidate Request is used to clear a specific subset of the address range from the ATC. Invalidate Requests are constrained to cover power of 2 multiple of 4096 byte pages.

The format of an Invalidation Request is shown in Figure 3-1.

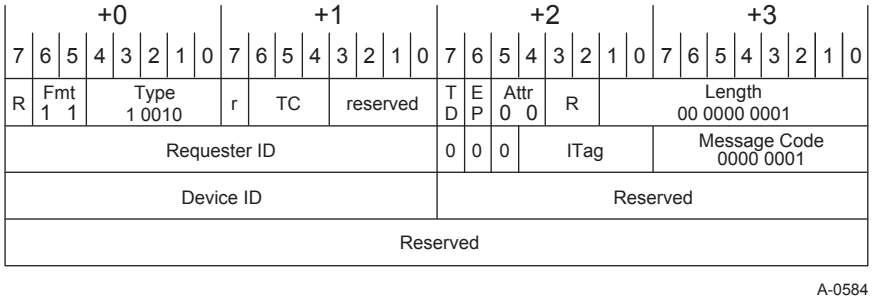

Figure 3-1: Invalidate Request Message

The Invalidate Request is a MsgD transaction with 64 bits of data. Invalidate Request messages may be sent in any TC. The ITag field is constrained to the values 0 to 31 and is used by the TA to uniquely identify requests it issues. A TA must ensure that once an ITag is used, it is not reused until released by the corresponding Invalidate Completes.

The TA may have a single pool of ITag values for all invalidates that it issues or it may have a pool for each Device ID or any other combination. An Endpoint with multiple ATC on different Functions must manage the ITags separately for each Requester ID.

The address range specified in an Invalidate Request may span one or more STU 4096-byte pages. Invalidation ranges are required to be naturally aligned and may not be smaller than STU 4096-byte pages.

Given ATPT designs are implementation-specific, Invalidate Completion timeouts and the associated error handling are treated as implementation-specific and therefore are outside the scope of this specification.

IMPLEMENTATION NOTE

Invalidate Completion Timeout. Choosing a single timeout value that works in all systems may not be possible. Implementations are encouraged to provide some latitude in the completion timeout ranges supported (See the *PCI Express Base Specification*).

The content of the payload is the un-translated address range to be invalidated. The payload format is shown in Figure 3-2.

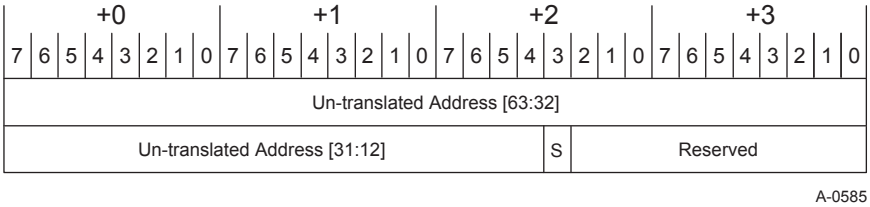

Figure 3-2: Invalidate Request Message Body

The S field is used to indicate if the range being invalidated is greater than 4096 bytes. Its meaning is the same as for the Translation Completion (see Section 2.3.1). An Un-translated Address value of all 1b's indicates that the entire cache should be invalidated.

3.2. Invalidate Complete

When a Function completes an Invalidate operation, it will send one or more Invalidate Complete messages to the TA. These messages will be tagged with information extracted from the Invalidate Request to enable the TA to associate the Invalidate Completes with the Invalidate Request.

The format of the Invalidate Complete Message is shown in Figure 3-3.

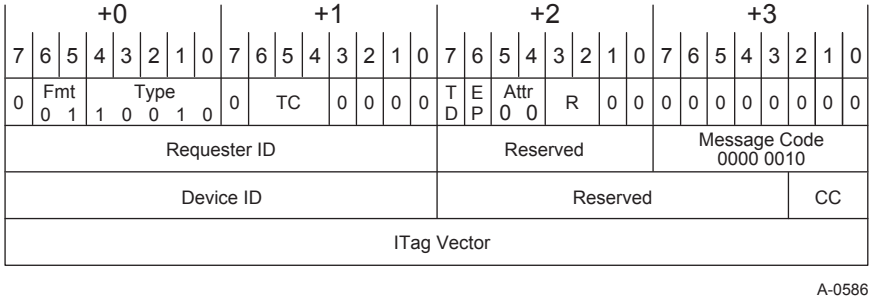

Figure 3-3: Invalidate Complete Message Format

The Invalidate Complete message is a Msg transaction routed by ID. The Requester ID field of the Invalidate Complete message is set to the Requester ID of the Function containing the ATC. The Device ID field of the Invalidate Complete is set to the Requester ID of the TA. The ATC may derive the Requester ID of the TA from the Requester ID field of the corresponding Invalidate Request. Alternatively, since the ATC is only associated with a single TA, the ATC may sample and store the Requester ID from the first Invalidate Request following a Fundamental Reset or FLR. Subsequent Invalidate Complete messages may use this value to set the Device ID field of Invalidate Complete messages.

The Completion Count (CC) field indicates the number of individual Invalidate Complete messages that will be sent for the associated Invalidate Request. Setting the CC field to 0 indicates that eight responses will be sent. The TA is responsible for collecting all the responses associated with a given Tag before considering the corresponding Invalidate Request to be complete.

Invalidate Complete messages may be sent on any TC, independent of the TC the originating Invalidate Request was received. This enables implementations to utilize the Invalidate Complete to push outstanding transactions to the TA to guarantee the required invalidation semantics are met Implementations that utilize a single upstream TC are required to send a single Invalidate Complete in the utilized TC.

The ITag Vector field is used to indicate which Invalidate Request has been completed. Each of the 32 possible ITag field values from the Invalidation Request is represented by a single bit in the ITag Vector field. The least significant bit (bit 0; i.e., the right-most bit in the schematic representation of the Invalidate Complete Message shown in Figure 3-3) of the ITag Vector field corresponds to the ITag field value of 0. The most significant bit (bit 31) of the ITag Vector field corresponds to the ITag field value of 31. Implementations are allowed to coalesce multiple Invalidate Completions by setting multiple ITag Vector bits in a single message provided the following conditions are met:

- The Invalidate Completions flow in the same TC.
- The Invalidate Completions have the same CC value.
- All fragments of an Invalidate Completion must have identical Request ID, CC, and ITag Vector fields.

When combining Invalidate Completions that are replicated in multiple TC (CC not equal to 1), each message of the replicated set may be combined independent of the others in the set.

Functions that do not support ATS or that have ATS services disabled will treat an Invalidate Request as UR. See the *PCI Base Specification* for futher details.

Functions supporting ATS are required to send an Invalidate Completion in response to a Invalidate Request independent of whether the BME bit is set or not. Note that the above conditions must be satisfied even when BME is set to a 0. The method for a device to achieve this is implementation dependent.

IMPLEMENTATION NOTE

When BME changes from set to clear no further memory requests should be queued. It is possible that queued write requests are present when BME is cleared. These requests could block an Invalidate Completion. These requests must be either sent or dropped. This will ensure that all outstanding write transactions that are potientially dependent upon the outstanding invalidation are complete.

3.3. Invalidation Completion Semantics

Before an ATC can return an Invalidate Complete for a given Invalidate Request, it must ensure the following conditions are satisfied:

- All new requests initiated by the Function will not utilize stale address translations.
- All outstanding read requests utilizing translated address matching the invalidated range have either completed or been tagged to be discarded (method to discard is implementation specific).
- All outstanding posted writes utilizing a translated address matching the invalidated range have been pushed to the TA. The ATC is required to send a copy of the Invalidate Complete message in each TC in which a posted write has been issued but not known to have been pushed to the TA. The CC field must be set to the same value in each copy of the Invalidate Complete message indicating number of copies sent. The TA is responsible for collecting all sent responses before considering the invalidation to be complete.

IMPLEMENTATION NOTE

Implied TC Flushing. When making the decision as to which TC to send Invalidate Completions, an ATC may infer, in an implementation specific manner, that an issued posted write has been pushed to the TA. For example, a Function that has sent a read transaction to a destination above

the TA and received its corresponding response may infer that any preceding posted writes issued in the same TC have been pushed to the TA.

3.4. Request Acceptance Rules

In accord with the request acceptance rules enumerated in the *PCI Express Base Specification*, a Function is not allowed to create a dependency in which the acceptance of a posted transaction is dependent upon the transmission of a posted transaction. Given Invalidate Requests and Invalidate Completes both are posted transactions, Functions must not make the acceptance of an Invalidate Request dependent upon the transmission of an Invalidate Complete. The method for achieving this is implementation specific.

IMPLEMENTATION NOTE

Invalidate Queue Depth. An ATC is only associated with a single TA. Each TA is limited to a total of 32 outstanding invalidations to any given ATC. This limits the number of outstanding Invalidation Requests active to a single ATC to 32. To avoid a post-to-post dependency, an ATC is required to implement sufficient queuing (32 entries) such that it is capable of holding all outstanding Invalidation Requests.

An ATC may choose to implement a maximally sized input queue holding Invalidate Requests. Alternatively, an ATC may choose to implement a maximally sized output queue holding Invalidate Responses. Note that queuing Invalidate Responses requires significantly less state per entry resulting in a potentially more efficient implementation than input queue buffering.

Note that the choice of whether to implement input queuing or output queuing (or a hybrid of both) has no impact on ensuring deadlock free behavior. But implementation choices with regard to queuing may have a significant impact on performance (see Section 3.5).

A Function with an ATS capability in its configuration space must be able to accept Invalidate Requests and send Invalidate Completions even if ATS is not enabled.

3.5. Invalidate Flow Control

Due to the variety of caching architectures and queuing strategies, implementations may vary greatly with respect to invalidation latency and throughput. It is possible that a TA may generate Invalidate Requests at a rate that exceeds the average ATC service rate. When this happens, the credit based flow control mechanisms will throttle the TA issue rate. A side effect of this is congestion spreading to other channels and Links though the credit based flow control mechanism. Depending on the frequency and duration of this congestion, performance may suffer. It is highly recommended that TA and its associated software implement higher level flow control mechanisms.

To assist with the implementation of Invalidate Flow Control, an ATC must publish the number of Invalidate Requests it can buffer before back pressuring the Link. This field applies to all invalidations serviced by the Function, independent of the size of the invalidation. This value is communicated in the Invalidate Queue Depth field in the ATS capability structure (see Section 4.1). A value of 0 0000b indicates that invalidate flow control is not necessary to this Function.

IMPLEMENTATION NOTE

Invalidate Flow Control. A Function may indicate that invalidate flow control is not required when one or more of the following is true:

1. The Function can handle invalidations at the maximum arrival rate of Invalidate Requests.
2. The Function will not or very rarely cause Link backpressure (performance loss is negligible).
3. The Function can fully buffer the maximum number of incoming invalidations without back pressuring the Link.

3.6. Invalidate Ordering Semantics

Invalidate Requests and Translation Completions may be sent using different TC and are, therefore, unordered with respect to each other (from the Link's perspective). An ATC must ensure that the proper invalidation behavior is maintained when an Invalidate Request bypasses a Translation Completion to an overlapping region.

An ATC must "snoop" its outstanding translation request queue against all arriving Invalidate Requests. When snooping a request for a N*STU sized translation, the ATC must snoop the range of addresses starting at the STU aligned region containing the specified address and ending (N-1) STU size pages later.

If an Invalidate Request overlaps the address range in an outstanding Translation Request, the Translation Request must be tagged as invalid and the results of its corresponding Translation Response must be discarded prior to transmission of the Invalidate Completion. If the Translation Response is received before the Invalidate Complete is sent, an implementation is free to issue requests utilizing the translation result provided the Invalidation Completion Semantics (see Section 3.3) are satisfied.

IMPLEMENTATION NOTE

Request Range Overlap in Invalidations. In the description above, N is the number of STU sized translations that were requested in the Translation Request. This is equal to (Length field in Translation Request)/2.

As an example:

STU is 00 0010b indicating 16384 bytes pages.

An outstanding Translation Request has a Length field of 00 0000 0100b indicating two translations which would cover a range of 32768 bytes.

The high-order 48 bits of the Translation Request are 0000 0FFF FFFFh.

The low-order 16 bits of the address in the request are 11xx xxxx xxxx xxxxb indicating that the translation request covers a range that overlaps a 32768-byte boundary (in fact, the request crosses a 16-TB boundary).

If two translations are returned, they would cover the two STU sized regions at 0000 0FFF FFFF C000h and 0000 1000 0000 0000h.

An Invalidate Request is received with the high-order 48 bits of 0000 1000 0000h and the low-order 16 bits of 0001 1xxx xxxx xxxxb.

The ATC must detect that a translation associated with a portion of the Translation Request is now invalidated and the Translation Completion associated with the invalidated region must be discarded (for simplification, the ATC is allowed to discard all of the Translation Completion).

It should be noted that processing of the Invalidates is simplified if Translation Requests do not cross alignment boundaries of the request. The Translation Request from the above example is not aligned to a 32768-byte boundary. If it were broken into two requests, it would be simpler to associate the range of the Invalidate Request with the address in the Translation Request. Breaking the Translation Requests into aligned requests is not a requirement.

3.7. Implicit Invalidation Events

The following events will cause the invalidation of all ATC entries:

- Fundamental Reset (all forms)
- Function Level Reset

No explicit Invalidate Complete is sent when the invalidation is implicit when these reset events occur.

4

4. Configuration

4.1. ATS Capability Structure

Each Function that supports ATS must have the ATS Capability structure in its configuration space.

Figure 4-1 details allocation of the register fields in the ATS Capability structure; Table 4-1 provides the respective field definitions.

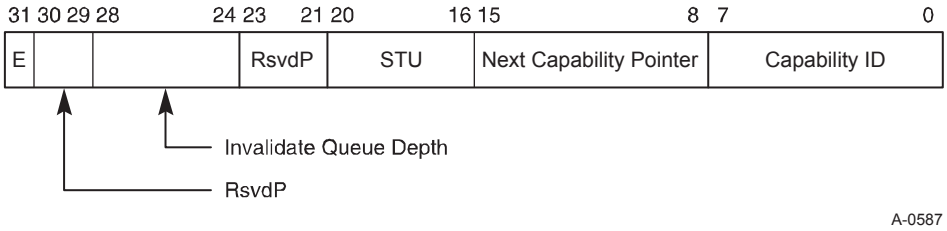

Figure 4-1: ATS Capability Structure

Table 4-1: ATS Capability Structure

Bit Location	Register Description	Attributes
7:0	**Capability ID** – Indicates the ATS Capability structure. This field must return a Capability ID of xxh **[TBD]** indicating that this is an ATS Capability structure.	RO
15:8	**Next Capability Pointer** – The offset to the next PCI Capability structure or 00h if no other items exist in the linked list of capabilities.	RO
20:16	**Smallest Translation Unit (STU)** – This value indicates to the Function the minimum number of 4096 byte blocks that will be indicated in a Translation Completion or Invalidate Requests. This is a power of 2 multiplier and the number of blocks is 2^STU. A value of 0 0000b indicates one block and a value of 1 1111b would indicate an 8-TB block. Default value is 0 0000b.	RW
28:24	**Invalidate Queue Depth** – The number of Invalidate Requests that the Function can accept before putting backpressure on the upstream connection. If 0 0000b, the Function can accept 32 Invalidate Requests.	RO
31	**Enable (E)** – When set, the Function is enabled to cache translations. Default value is 0b.	RW

Acknowledgements

The following persons were instrumental in the development of the ATS Specification:[2]

Andrew Gruber, ATI Corporation

Mark Hummel, Advanced Micro Devices, Inc.

Michael Krause, Hewlett-Packard Corporation

Brian Langendorf, Nvidia Corporation

Rajesh Sankaran, Intel Corporation

David Wooten, Microsoft Corporation

William Wu, Broadcom Corporation

[2] Company affiliation listed is at the time of specification contributions.